☑ FASE 4 — Diseño de Arquitectura Objetivo (Microservicios) (Sin modificar código)

Proyecto: Encuestas Espagueti
Repositorio: Semana-2
Base: Hallazgos Fase 2 + Propuesta Fase 3
Autor:

- Juan Sebastian Osorio Fierro
- Daniel Steve Fontalvo Matiz
- Leonado Fabio Perez Bermudez
- Juan Camilo Cruz Pardo **Fecha:** 2026-02-23

> **Restricción:** Esta fase es documental. **No se modifica el código**.

---

1. Objetivo de la Fase 4

Diseñar la arquitectura objetivo del sistema para evolucionar desde un monolito “espagueti” hacia una solución escalable y mantenible, mediante:

- Descomposición por **bounded contexts**.
- Definición de **microservicios** con responsabilidades claras.
- Propuesta de infraestructura (Gateway, Config, Discovery, Observabilidad).
- Trazabilidad con decisiones arquitectónicas mediante **ADRs**.
- Soporte visual mediante diagramas **C4**.

---

2. Principios arquitectónicos adoptados

1. **Separación de responsabilidades:** cada servicio tiene un propósito claro (alta cohesión).
2. **Data ownership:** cada microservicio es dueño de su base de datos (evita acoplamiento por JOIN).
3. **Contratos explícitos:** APIs definidas con contratos tipados (DTOs / interfaces).
4. **Configuración externa:** secretos y URLs fuera del código (env / config server).
5. **Observabilidad:** logs consistentes y trazabilidad para operación.

---

3. Bounded Contexts (dominio) propuestos

El dominio “Encuestas” se divide en dos contextos principales:

3.1 Survey Context (Gestión de encuestas)

Responsable de:

- Crear encuestas
- Listar encuestas
- Consultar encuesta por ID

Relaciona directamente los endpoints actuales:

- `POST /crear`
- `GET /encuestas`
- `GET /encuesta/{id}`

3.2 Voting Context (Votación y resultados)

Responsable de:

- Registrar votos por encuesta
- Exponer resultados (si aplica)

Relaciona el endpoint actual:

- `POST /votar`

---

4. Microservicios propuestos (Arquitectura Objetivo)

4.1 survey-service

Responsabilidad: CRUD y consultas de encuestas (sin lógica de votos).

Endpoints sugeridos:

- `POST /surveys` (equivalente conceptual a `/crear`)
- `GET /surveys` (equivalente a `/encuestas`)
- `GET /surveys/{id}` (equivalente a `/encuesta/{id}`)

Datos que administra (data ownership):

- Encuesta (id, pregunta, ...)

Base de datos:

- PostgreSQL (schema o DB propia: `surveys`)

4.2 voting-service

Responsabilidad: registrar votos y administrar resultados.

Endpoints sugeridos:

- `POST /votes` (equivalente conceptual a `/votar`)
- `GET /results/{surveyId}` (opcional, si se requiere endpoint de resultados)

Datos que administra:

- Voto (id, surveyId, tipoVoto, timestamp) **o** contadores agregados por encuesta

Base de datos:

- PostgreSQL (schema o DB propia: `votes`)

> Nota: se puede optar por:
>
> - **Modelo A (agregado):** guardar contadores si/no por encuesta en voting-service.
> - **Modelo B (eventos):** guardar votos individuales y calcular resultados. En este documento se deja como decisión formal en ADR.

---

5. Componentes transversales (Infraestructura)

5.1 API Gateway

Función:

- Punto de entrada único para frontend
- Enrutamiento hacia survey-service y voting-service
- Control de CORS
- (Opcional) Autenticación/JWT si evoluciona el sistema
- (Opcional) Rate limiting

5.2 Configuración centralizada

Función:

- Variables por entorno (DEV/QA/PROD)
- Secretos (DB_URL, DB_USER, DB_PASS)

Se puede implementar con:

- Config Server (opcional)
- Variables de entorno y Docker Compose (mínimo viable)

5.3 Service Discovery (opcional)

- Eureka/Consul si se requiere descubrimiento dinámico

5.4 Observabilidad (recomendado)

- Logs estructurados por servicio
- Trazas (si el curso lo exige, se menciona como propuesta)

---

6. Diagramas C4 (Fase 4)

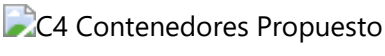
6.1 C4 — Contexto (Nivel 1)


C4 Contexto

6.2 C4 — Contenedores (Nivel 2) — Estado Actual


C4 Contenedores Actual

6.3 C4 — Contenedores (Nivel 2) — Propuesto (Microservicios)



> **Fuente del diagrama:** ver carpeta `source/` (PlantUML .puml)

---

7. Contratos de comunicación entre servicios

7.1 Comunicación sincrónica (REST)

- Frontend → Gateway → Servicios
- Gateway enruta por path:
  - `/surveys/**` → survey-service
  - `/votes/**` y `/results/**` → voting-service

7.2 Comunicación asincrónica (opcional)

Si se usa broker:

- voting-service publica evento `VoteCast`
- survey-service podría consumir para analítica (opcional)

---

8. Tabla de responsabilidades (resumen)

Servicio	Responsabilidad	Endpoints	DB
API Gateway	Entrada única, routing, CORS	Proxy	No
survey-service	Gestión encuestas	/surveys	Sí (surveys)
voting-service	Votos y resultados	/votes, /results	Sí (votes)

---

9. ADRs — Decisiones Arquitectónicas Fase 4

ADR-006 — Descomposición por bounded contexts (Survey y Voting)

Estado: Aceptada
Contexto: El monolito mezcla encuestas + votos + SQL + configuración en una clase.
Decisión: Separar el dominio en 2 microservicios: `survey-service` y `voting-service`.
Alternativas:

1. Mantener monolito por capas
2. Microservicios por cada tabla
3. Microservicios por bounded context (**seleccionada**)
   Consecuencias:

- ☑ Alta cohesión por servicio
- ☑ Permite escalado independiente
- ⚠ Mayor complejidad operativa

ADR-007 — Data ownership: DB por microservicio

Estado: Aceptada
Contexto: Compartir DB entre servicios aumenta acoplamiento y dependencia por JOIN.
Decisión: Cada servicio gestiona su propia DB/schema.
Alternativas:

1. DB compartida
2. DB por servicio (**seleccionada**)
   Consecuencias:

- ☑ Menor acoplamiento
- ☑ Independencia de despliegue
- ⚠ Consistencia eventual (si hay datos cruzados)

ADR-008 — API Gateway como punto de entrada único

Estado: Aceptada
Contexto: El frontend no debería depender de múltiples endpoints/hosts ni manejar CORS disperso.
Decisión: Introducir Gateway para enrutar a microservicios.
Alternativas:

1. Frontend consume servicios directamente
2. Gateway (**seleccionada**)
   Consecuencias:

- ☑ CORS, routing y seguridad centralizados
- ☑ Evolución más ordenada
- ⚠ Componente adicional a operar

ADR-009 — Estrategia de resultados de votación (agregado vs votos individuales)

Estado: Propuesta (pendiente)
Contexto: El monolito actual mantiene contadores `si_count/no_count` en la tabla `encuestas`.
Decisión propuesta: Mantener resultados en `voting-service` para desacoplar dominio de encuestas del dominio de votos.
Alternativas:

- A) Guardar contadores en voting-service (más simple)
- B) Guardar votos individuales + cálculo (más robusto) **Consecuencias:**
- A) ☑ simple / ⚠ menos flexible
- B) ☑ analítica / ⚠ mayor costo de almacenamiento y cómputo

---

10. Plan de transición (documental) Monolito → Microservicios

1. **Paso 1:** Aplicar arquitectura en capas (Fase 3) en el monolito (sin cambiar funcionalidad).

2. **Paso 2:** Extraer survey-service (endpoints de encuestas) manteniendo el contrato.
3. **Paso 3:** Extraer voting-service (votos/resultados).
4. **Paso 4:** Introducir API Gateway y redirigir frontend a un único punto.
5. **Paso 5:** Separar DBs por servicio y definir estrategia de sincronización (si aplica).

---

11. Conclusión Fase 4

La arquitectura propuesta organiza el dominio en bounded contexts, habilita escalabilidad y reduce el acoplamiento estructural del monolito.
El uso de microservicios + gateway + configuración externa establece una base sólida para evolución del sistema, manteniendo trazabilidad mediante ADRs y soporte visual con C4.